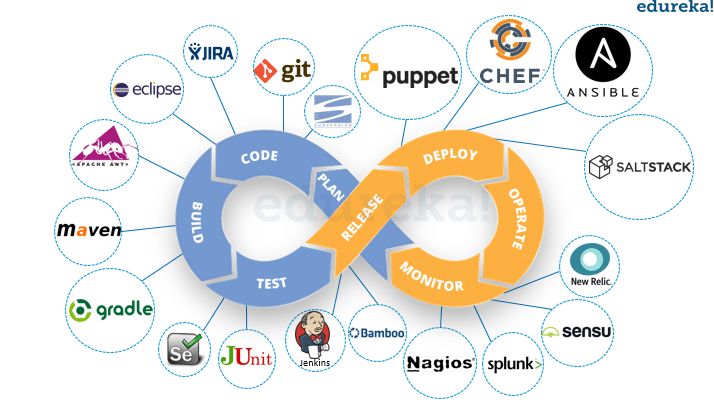
## **What is DevOps?**



* The term DevOps is a combination of two words namely Development and Operations. DevOps is a practice that allows a single team to manage the entire application development life cycle, that is, development, testing, deployment, and monitoring.
* The ultimate goal of DevOps is to decrease the duration of the system’s development life cycle while delivering features, fixes, and updates frequently in close synchronization with business objectives.
* DevOps is a software development approach with the help of which you can develop superior quality software quickly and with more reliability. It consists of various stages such as continuous development, continuous integration, continuous testing, continuous deployment, and continuous monitoring.

## DevOps Stages and Tools

As mentioned earlier, the various stages such as continuous development, continuous integration, continuous testing, continuous deployment, and continuous monitoring constitute the DevOps Life cycle. Now let us have a look at each of the stages of DevOps life cycle one by one.

**Stage – 1: Continuous Development**

**Tools Used: Git, SVN, Mercurial, CVS**

* This is the phase that involves ‘planning‘ and ‘coding‘ of the software. You decide the project vision during the planning phase and the developers begin developing the code for the application.
* There are no [DevOps tools](https://www.edureka.co/blog/devops-tools" \t "_blank) that are required for planning, but there are a number of tools for maintaining the code.
* The code can be in any language, but you maintain it by using Version Control tools. This process of maintaining the code is known as Source Code Management.
* After the code is developed, then you move to the Continuous Integration phase.

**Stage – 2: Continuous Integration**

**Tools: Jenkins, TeamCity, Travis**

* This stage is the core of the entire DevOps life cycle. It is a practice in which the developers require to commit changes to the source code more frequently. This may be either on a daily or weekly basis.
* You then build every commit and this allows early detection of problems if they are present. Building code not only involves compilation but it also includes code review, unit testing, integration testing, and packaging.
* The code supporting new functionality is [continuously integrated](https://www.edureka.co/blog/continuous-integration/) with the existing code. Since there is a continuous development of software, you need to integrate the updated code continuously as well as smoothly with the systems to reflect changes to the end-users.
* In this stage, you use the tools for building/ packaging the code into an executable file so that you can forward it to the next phases.

**Stage – 3: Continuous Testing**

**Tools: Jenkins, Selenium TestNG, JUnit**

* This is the stage where you test the developed software continuously for bugs using automation testing tools. These tools allow QAs to test multiple code-bases thoroughly in parallel to ensure that there are no flaws in the functionality. In this phase, you can use Docker Containers for simulating the test environment.
* [**Selenium**](https://selenium.dev/) is used for automation testing, and the reports are generated by [TestNG](https://www.edureka.co/blog/selenium-webdriver-tutorial" \t "_blank). You can automate this entire testing phase with the help of a Continuous Integration tool called Jenkins.
* Suppose you have written a selenium code in Java to test your application. Now you can build this code using ant or maven. Once you build the code, you then test it for User Acceptance Testing (UAT). This entire process can be automated using [Jenkins](https://www.edureka.co/blog/jenkins-tutorial/).

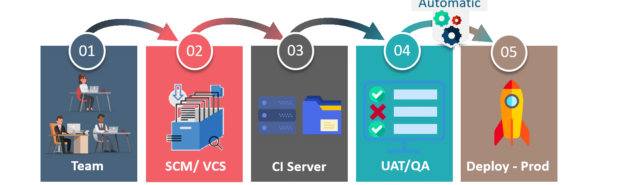
**Stage – 4: Continuous Deployment**

**Tools Used:**

**Configuration Management – Chef, Puppet, Ansible**

**Containerization – Docker, Vagrant**

* This is the stage where you deploy the code on the production servers. It is also important to ensure that you correctly deploy the code on all the servers. Before moving on, let us try to understand a few things about Configuration management and [Containerization tools](https://www.edureka.co/blog/docker-tutorial). These set of tools here help in achieving Continuous Deployment (CD).



* + [Configuration Management](https://www.edureka.co/blog/what-is-puppet/) is the act of establishing and maintaining consistency in an application’s functional requirements and performance. Let me put this in easier words, it is the act of releasing deployments to servers, scheduling updates on all servers and most importantly keeping the configurations consistent across all the servers.
  + Containerization tools also play an equally crucial role in the deployment stage. The containerization tools help produce consistency across Development, Test, Staging as well as Production environments. Besides this, they also help in scaling-up and scaling-down of instances swiftly.

**Stage – 5: Continuous Monitoring**

**Tools Used: Splunk, ELK Stack, Nagios, New Relic**

* This is a very critical stage of the DevOps life cycle where you continuously monitor the performance of your application. Here you record vital information about the use of the software. You then process this information to check the proper functionality of the application. You resolve system errors such as low memory, server not reachable, etc in this phase.
* This practice involves the participation of the Operations team who will monitor the user activity for bugs or any improper behavior of the system. The Continuous Monitoring tools help you monitor the application’s performance and the servers closely and also enable you to check the health of the system proactively.

## **Who is a DevOps Engineer?**

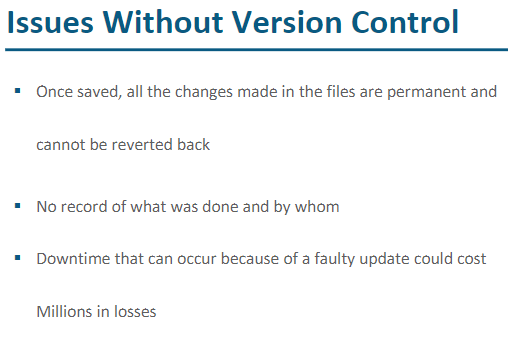
DevOps Engineer is somebody who understands the Software Development Lifecycle and has the outright understanding of various automation tools for developing digital pipelines (CI/ CD pipelines).

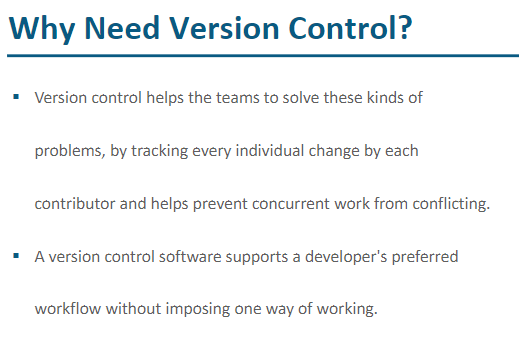
DevOps Engineer works with developers and the IT staff to oversee the code releases. They are either developers who get interested in deployment and network operations or sysadmins who have a passion for scripting and coding and move into the development side where they can improve the planning of test and deployment.

**GIT :**

Git is a free, open source distributed version control system tool designed to handle everything from small to very large projects with speed and efficiency.

Git is primarily used to manage your project, comprising a set of code/text files that may change.



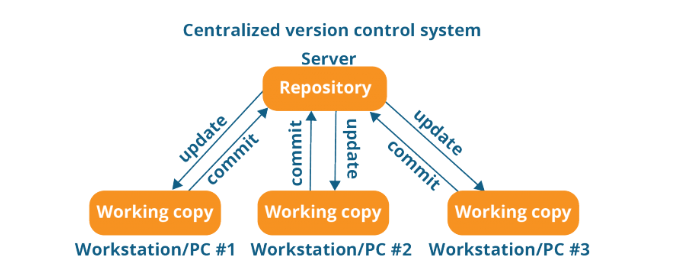


## **There are two types of VCS:**

* Centralized Version Control System (CVCS)
* Distributed Version Control System (DVCS)

### **Centralized VCS**

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server.



The repository in the above diagram indicates a central server that could be local or remote which is directly connected to each of the programmer’s workstation.

Every programmer can extract or **update** their workstations with the data present in the repository or can make changes to the data or **commit** in the repository. Every operation is performed directly on the repository.

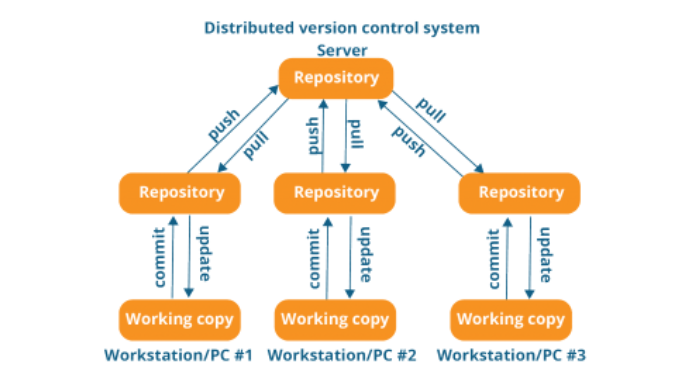
Even though it seems pretty convenient to maintain a single repository, it has some major drawbacks. Some of them are:

* It is not locally available; meaning you always need to be connected to a network to perform any action.
* Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.

### **Distributed VCS**

These systems do not necessarily rely on a central server to store all the versions of a project file.

In Distributed VCS, every contributor has a local copy or “clone” of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.



As you can see in the above diagram, every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive. They can commit and update their local repository without any interference.

They can update their local repositories with new data from the central server by an operation called “**pull**” and affect changes to the main repository by an operation called “**push**” from their local repository.

### ***What is the difference between Git vs GitHub?***

Git is just a version control system that manages and tracks changes to your source code whereas GitHub is a cloud-based hosting platform that manages all your Git repositories.

**GIT Commands**

### ***Git config command :*** This command configures the user. The Git config command is the first and necessary command used on the Git command line. This command sets the author name and email address to be used with your commits. Git config is also used in other scenarios.

$ git config --global user.name "user.name"

$ git config --global user.email "user.email"

### ***Git Init command*** This command is used to create a local repository. The init command will initialize an empty repository

$ git init

### ***Git clone command :*** This command is used to make a copy of a repository from an existing URL. If I want a local copy of my repository from GitHub, this command allows creating a local copy of that repository on your local directory from the repository URL.

$ git clone URL

### ***Git add command :*** This command is used to add one or more files to staging (Index) area.

To add one file **$ git add Filename**

To add more than one file **$ git add\***

**Why we need data into staging area? :** The git add command adds a change in the working directory to the staging area. **It tells Git that you want to include updates to a particular file in the next commit**. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit .

### ***Git commit command***

$ git commit -m " Commit Message"

### ***Git status command* :** The status command is used to display the state of the working directory and the staging area. It allows you to see which changes have been staged, which haven't, and which files aren?t being tracked by Git. It does not show you any information about the committed project history. For this, you need to use the git log. It also lists the files that you've changed and those you still need to add or commit.

$ git status

***Git push Command :*** It is used to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repo. It's the complement to git fetch, but whereas fetching imports commits to local branches on comparatively pushing exports commits to remote branches. Remote branches are configured by using the git remote command. Pushing is capable of overwriting changes, and caution should be taken when pushing.

Git push command can be used as follows.

### **Git push origin master** This command sends the changes made on the master branch, to your remote repository.

$ git push [variable name] master

**Git push –all** This command pushes all the branches to the server repository.

$ git push --all

### **Git pull command** Pull command is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory.

$ git pull URL

### **Git Branch Command** This command lists all the branches available in the repository.

$ git branch

### **Git Merge Command** This command is used to merge the specified branch?s history into the current branch.

$ git merge BranchName

### **Git log Command** This command is used to check the commit history.

$ git log

### **Git remote Command** Git Remote command is used to connect your local repository to the remote server. This command allows you to create, view, and delete connections to other repositories. These connections are more like bookmarks rather than direct links into other repositories. This command doesn't provide real-time access to repositories.

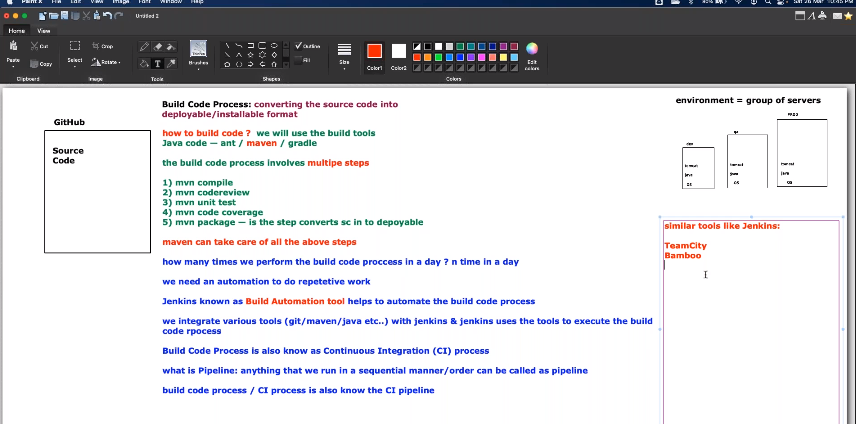
**Maven**

**Question : What is Maven ?**

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

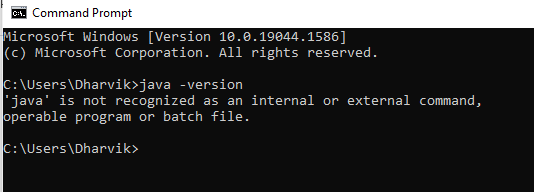
**Once source code is available in GIT then we need some Build process tool to convert source code into deployable code.**

Maven is build code process tool (converting source code into deployable intallable format)

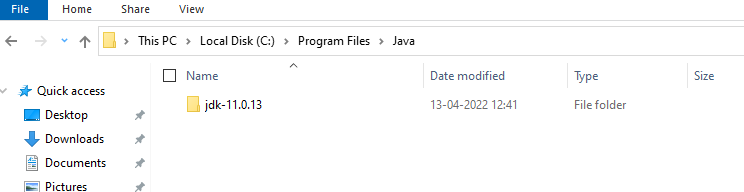


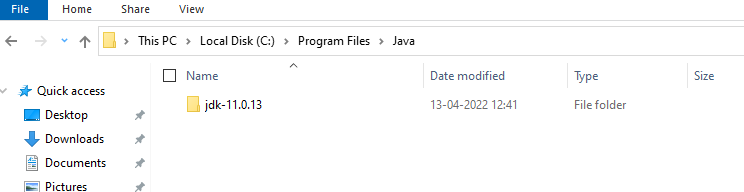
**Steps to create maven project in eclipse**

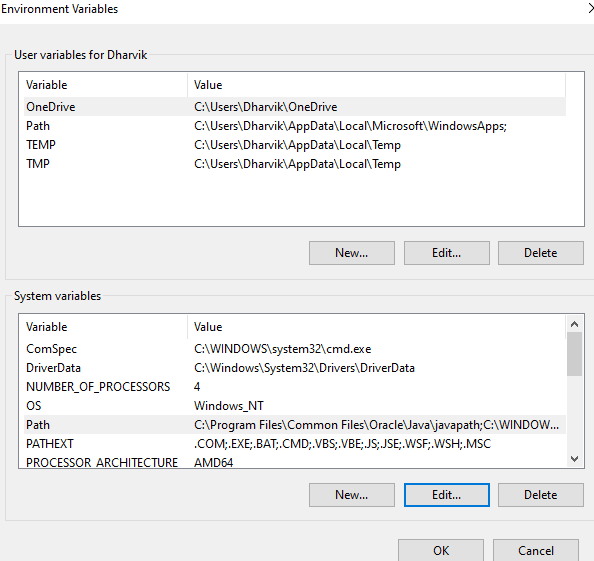
First install JDK and set environment variables, before installing chel Java in command prompt if it it already installed or not **(Command used is Java –version)**



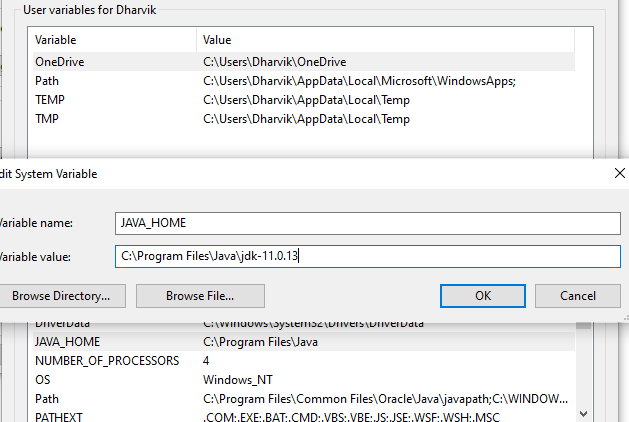
Now download JDK and install and after installation set environment variables

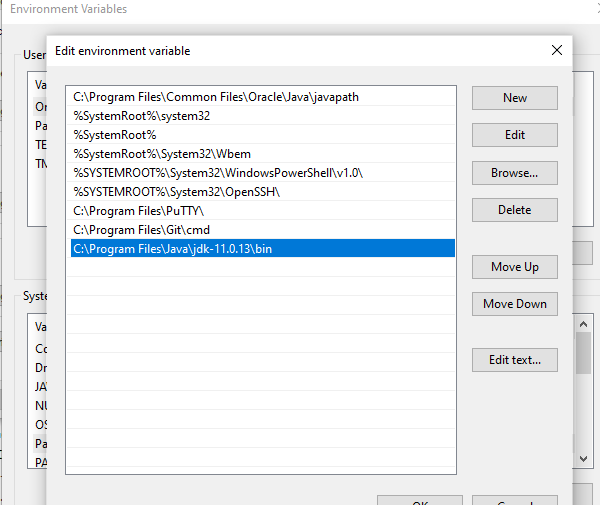




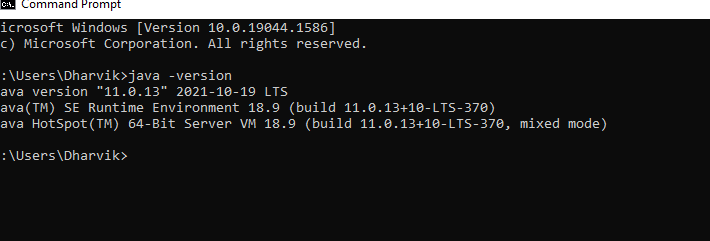


Add Java \_home and Path in system variables



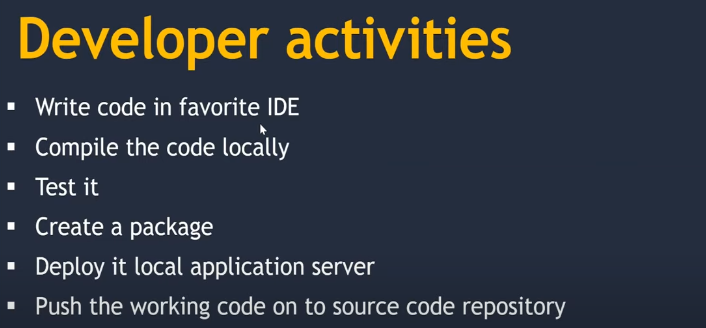


As Path is set now, check Java from command prompt



Now install Eclipse and create new Java project with Maven , If we install then by default maven will be installed in Eclipse

**Day to day activities of a developer:**



While developing the code , developer will use some third party libraries and packages by downloading them.

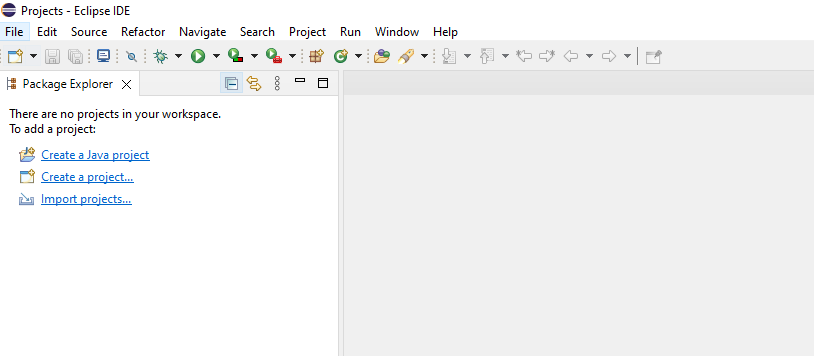
So while pushing the code into source code repository developer can push only the code and not third party packages so if some one need to use this code from reposirtoy then they might need to download packaged again to build which will be difficult.

**To avoid this if we use maven then maven will download all the third party packages and build the code.**

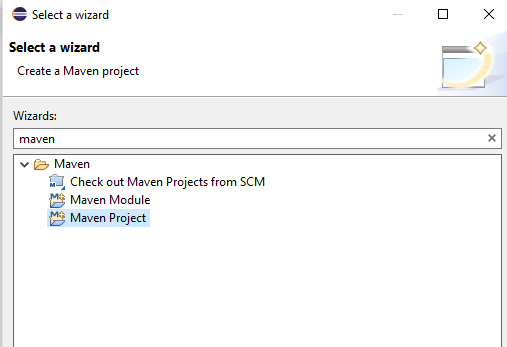
**Maven will help developer to develop, build, test and deploy the code efficiently.**

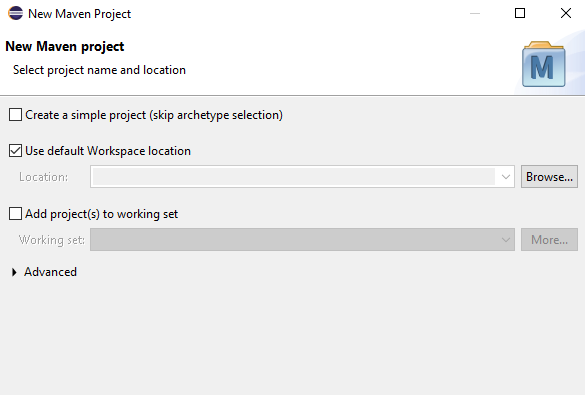


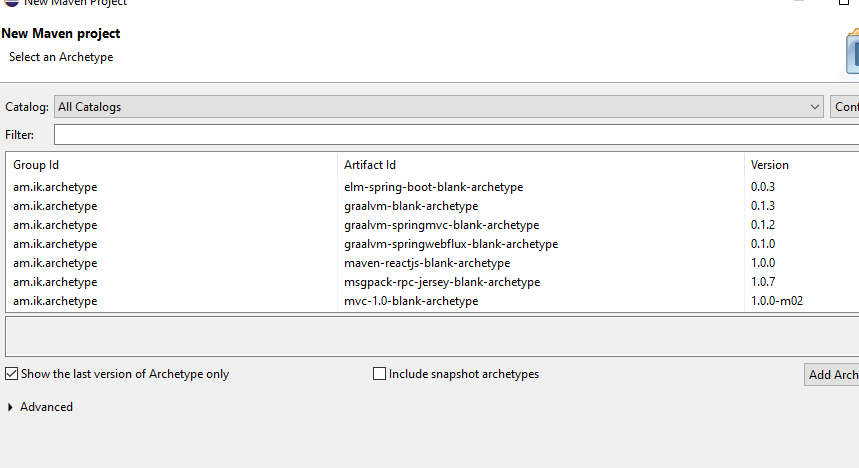
**Creating Maven Project in Eclipse**

**b**

**Go to File > New > Other ( As maven option is not present)**

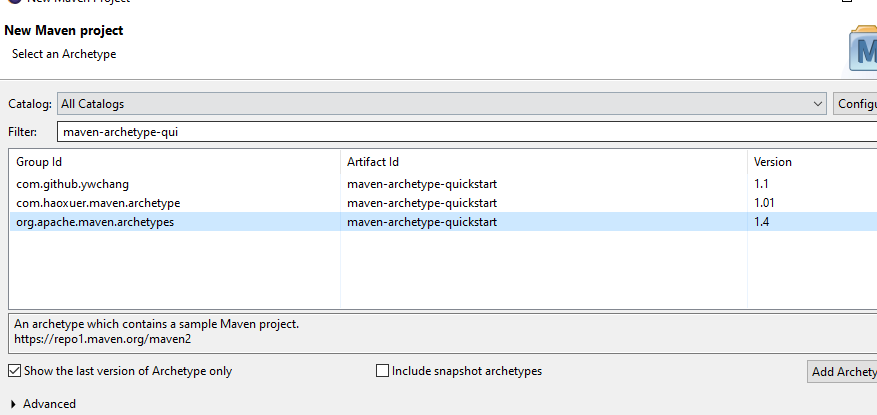




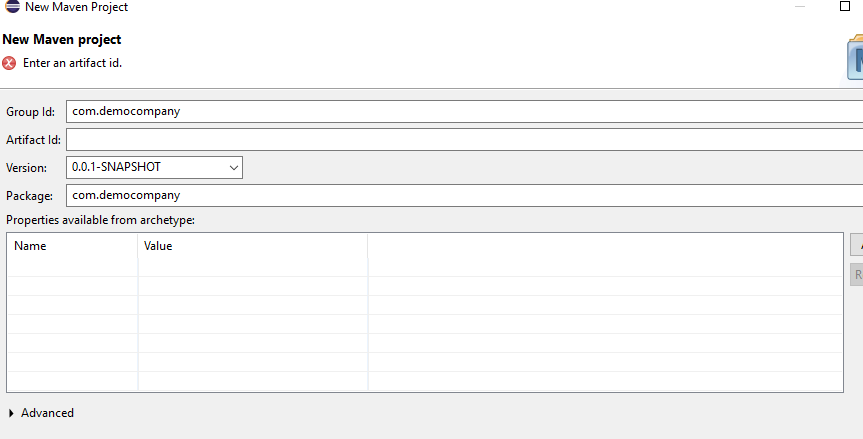


Archetype is **a Maven project templating toolkit**. An archetype is defined as an original pattern or model from which all other things of the same kind are made.

Archetype will **help authors create Maven project templates for users, and provides users with the means to generate parameterized versions of those project templates**.

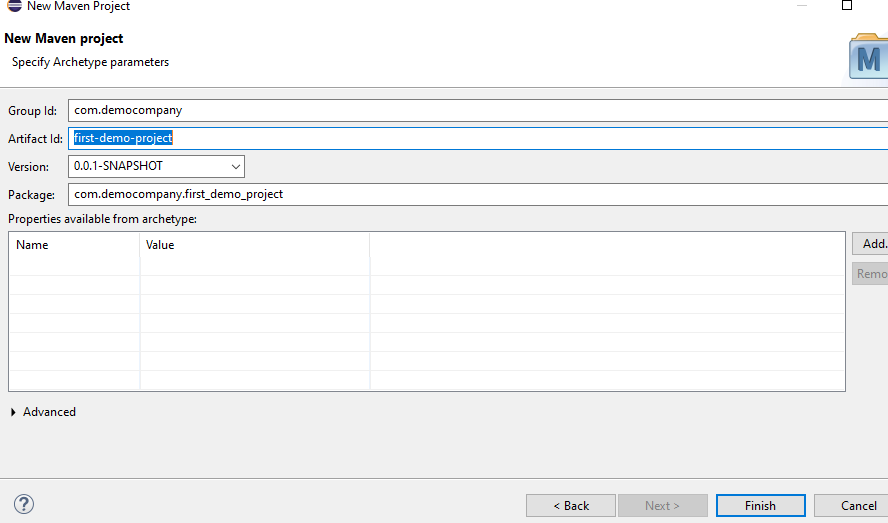


**Select Next after maven-archetype**

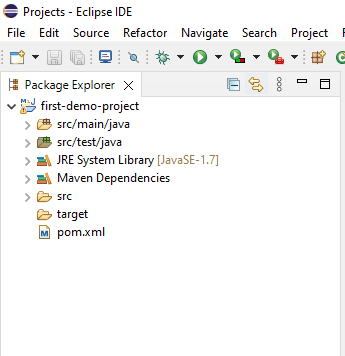


Group id , Artifact id and Version are the maven coordinates .

Maven coordinates **identify uniquely a project, a dependency, or a plugin defined in POM**.



**Click Finish**



**Project is created successfully,**

**src/main/java –** This is where we need to write the code

**src/main/test –** This is where we need to write test cases

**pom.xml – project object model** which tells about the dependancies of this project,outcome of the project, maven coordinates, plugins

**target** – once the build process is completed then output will be stored here.

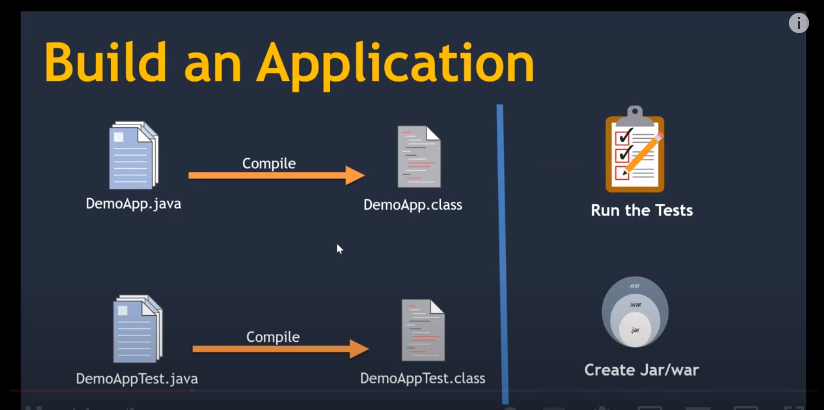
To build any application we need compile the code first and then we need to run the test cases



A JAR (Java ARchive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file for distribution.

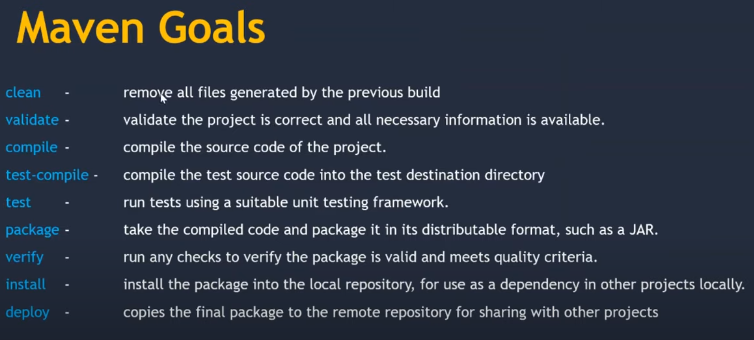
An EAR file is a critical piece in deploying a service application to a production server. An enterprise archive (EAR) file is a compressed file that contains the libraries, enterprise beans, and JAR files that the application requires for deployment.

In [software engineering](https://en.wikipedia.org/wiki/Software_engineering), a WAR file (Web Application Resource[[1]](https://en.wikipedia.org/wiki/WAR_(file_format)#cite_note-1) or Web application ARchive[[2]](https://en.wikipedia.org/wiki/WAR_(file_format)" \l "cite_note-2)) is a file used to distribute a collection of [JAR](https://en.wikipedia.org/wiki/JAR_(file_format))-files, [JavaServer Pages](https://en.wikipedia.org/wiki/JavaServer_Pages" \o "JavaServer Pages), [Java Servlets](https://en.wikipedia.org/wiki/Java_Servlet), [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) [classes](https://en.wikipedia.org/wiki/Class_(file_format)), [XML](https://en.wikipedia.org/wiki/XML) files, tag libraries, static web pages ([HTML](https://en.wikipedia.org/wiki/HTML) and related files) and other resources that together constitute a [web application](https://en.wikipedia.org/wiki/Web_application).

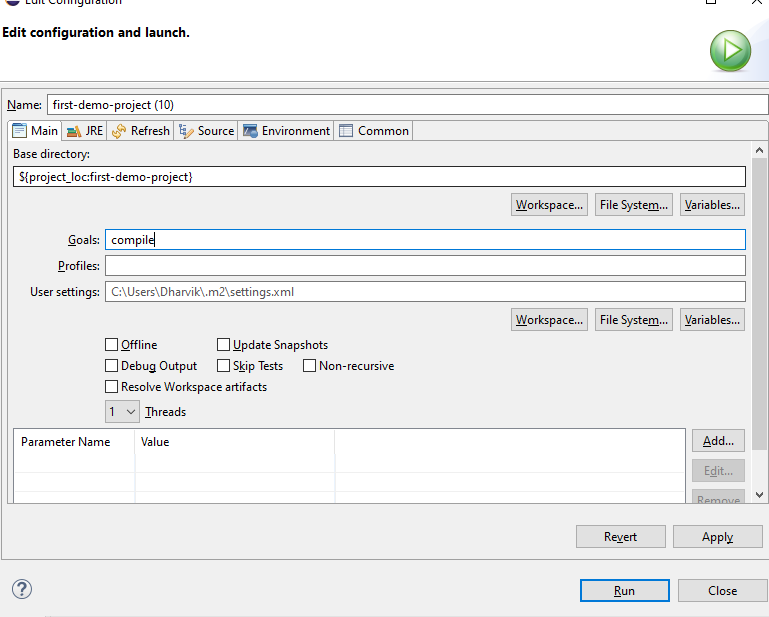


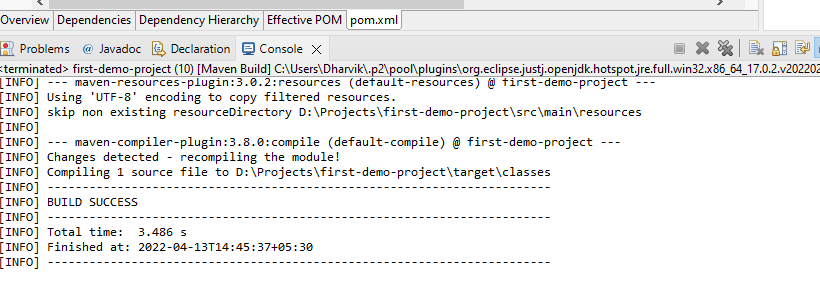
**Now to build the code , right click on the project > Run As > maven build ( if we select this one then this will run previously run goal) and if we select Maven build…(this will run allow user to enter new goal)**

**Maven goals – maven goals are set in the build life cycle format**

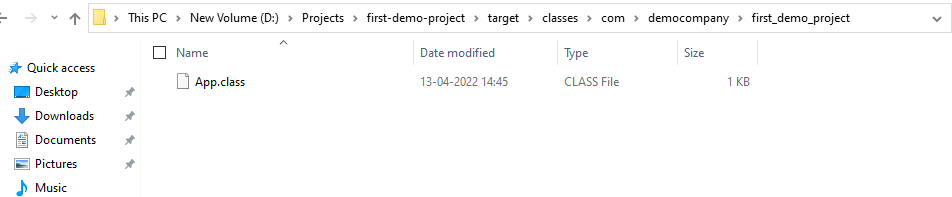


**If we execute verify goal then all the previous goals before verify will be executed, so if we use deploy goal then it will execute all the default life cycle goals**



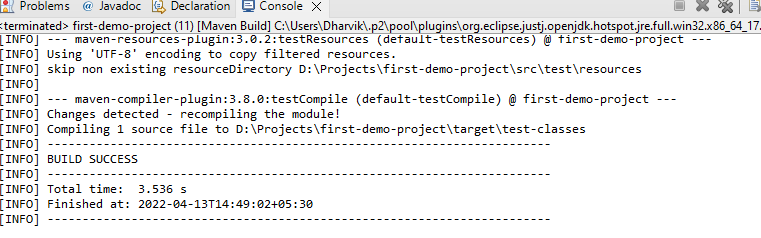


**In the console we can see build success and we can see the goal executed is default –compile**

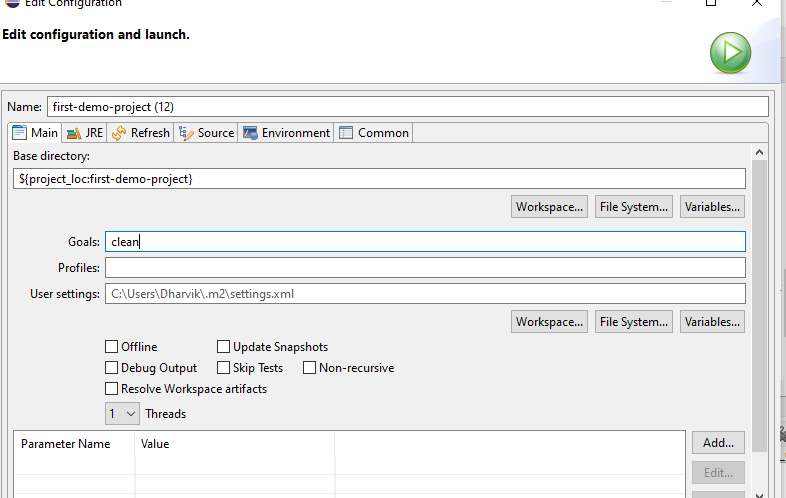


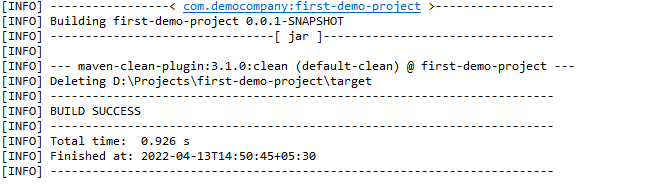
**After build we can see the app.class file in target folder which was not there before**

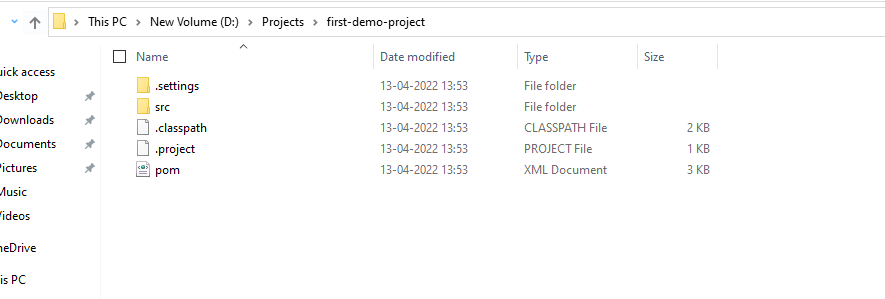
**Now execute test compile goal**



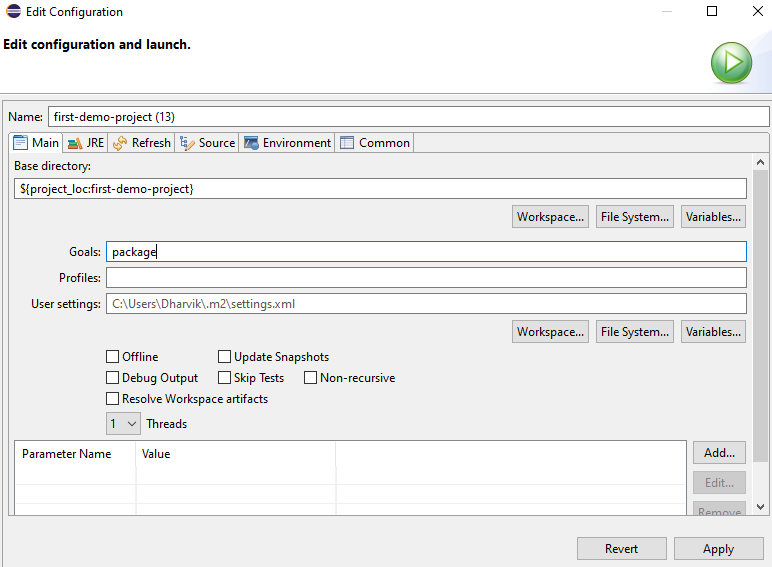
**Now run clean goal which will clear target folder**

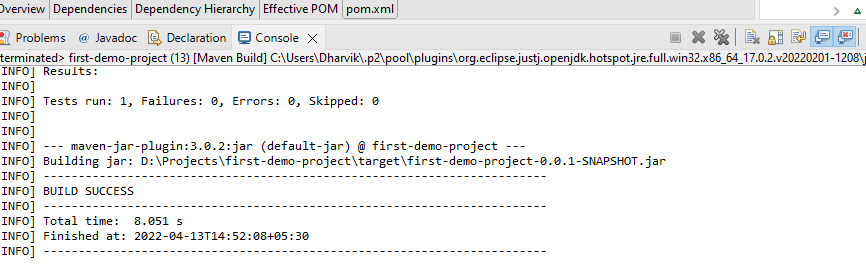




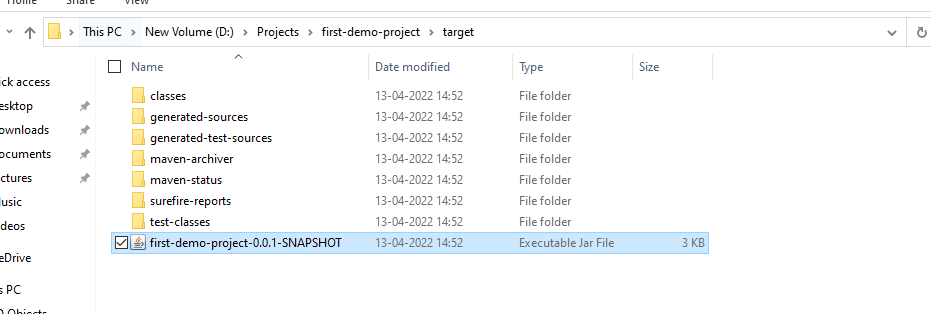


**Now run package folder which will create jar file**





**In the console we can jar file and also we can see in the target folder**



**POM.xml:** A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects

Pom.xml will the artifact id details of the project which we created

Also pom.xml will have the dependences which required for the project

For example in our project we need Junit to run the tests so Junit dependency is added and we can see the same under maven dependancies

<dependencies>

<dependency>

<groupId>junit</groupId>

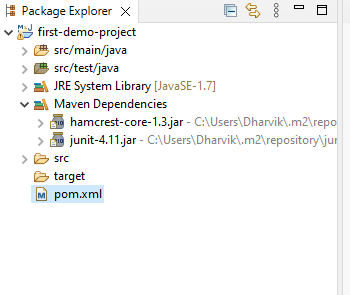
<artifactId>junit</artifactId>

<version>4.11</version>

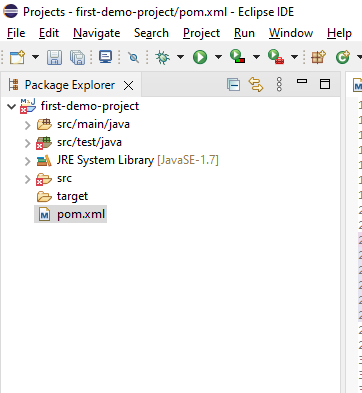
<scope>test</scope>

</dependency>

</dependencies>

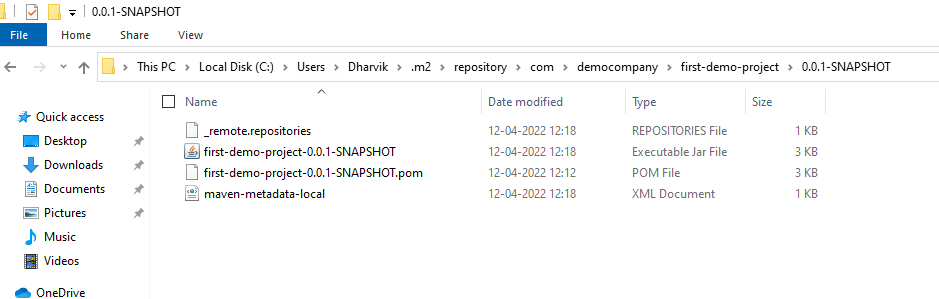


Now if I remove Junit dependency then we cannot see maven dependency



there is no maven dependancies as I removed from pom.xml

Whenever we install (goal is install) the package in local then it will be create under local repository under users/.m2 directory



**Deploy goal:** This goal is used deploy the code in central repository .

**Src/main/java, src/test and pom.xml -- These three are enough to push the code into central repository**

**“/” is the root of the system.** **“/root” is the home directory of the user root**. Typically “/root” would be the administrator of the system. You could however give the administrator a completely other or no home directory if you so wish and ditch “/root” all together if even present on your system.

**Settings .xml in maven**

The settings element in the settings.xml file contains elements used to define values which configure Maven execution in various ways, like the pom.xml, but should not be bundled to any specific project, or distributed to an audience. These include values such as the local repository location, alternate remote repository servers, and authentication information.

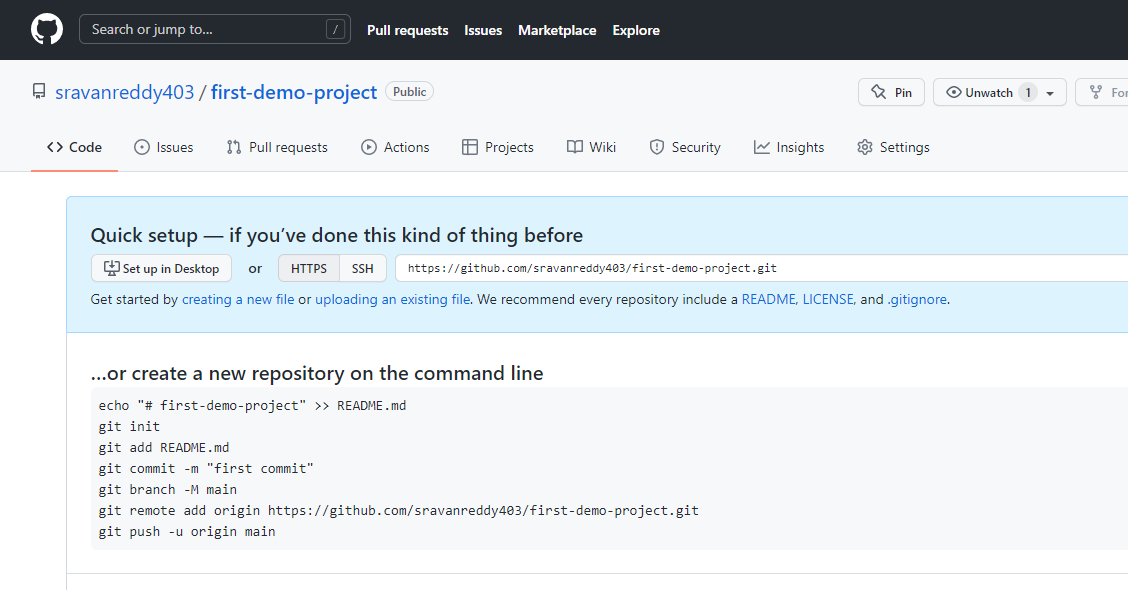
There are two locations where a settings.xml file may live:

* The Maven install: ${maven.home}/conf/settings.xml
* A user's install: ${user.home}/.m2/settings.xml

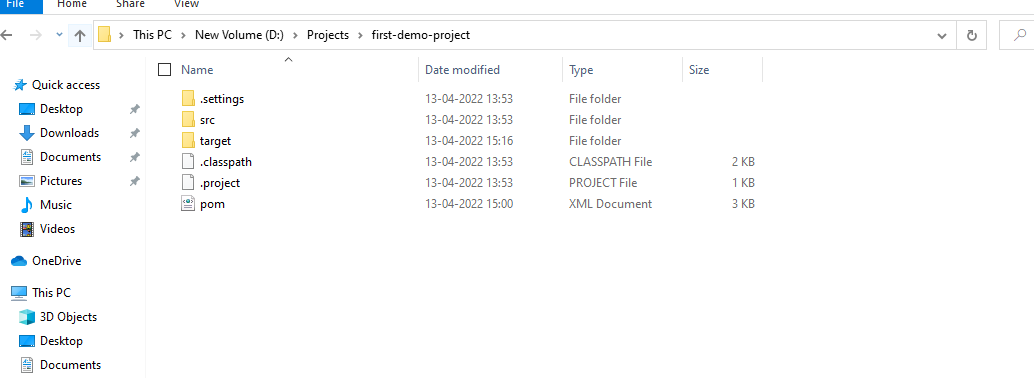
The former settings.xml are also called global settings, the latter settings.xml are referred to as user settings. If both files exists, their contents gets merged, with the user-specific settings.xml being dominant.

Tip: If you need to create user-specific settings from scratch, it's easiest to copy the global settings from your Maven installation to your ${user.home}/.m2 directory. Maven's default settings.xml is a template with comments and examples so you can quickly tweak it to match your needs.

**Create Repository in github**

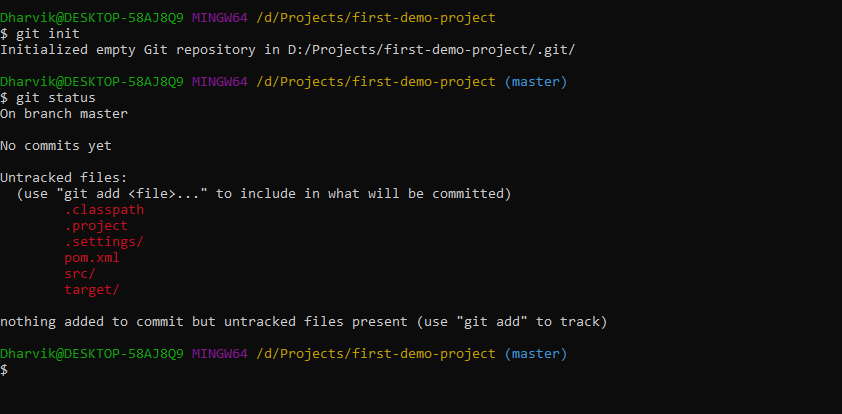


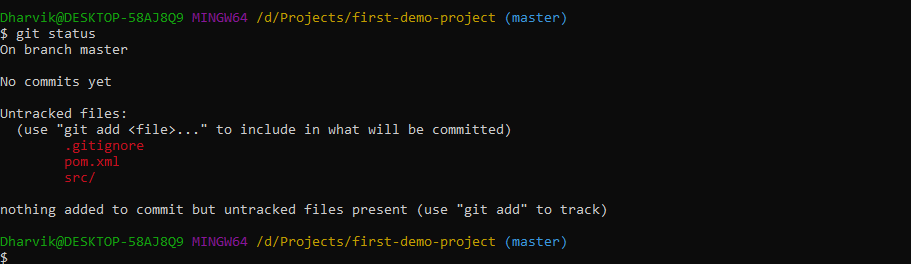
To push the code from local to central first convert local folder as git repo folder and then move to central folder

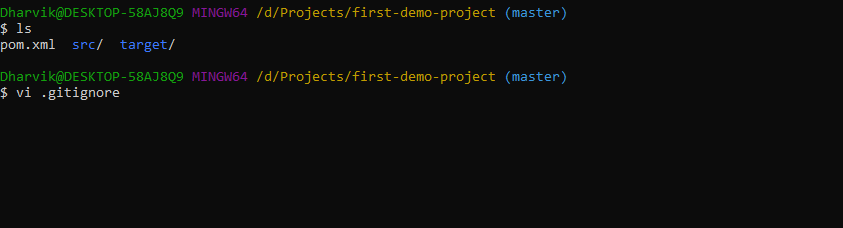


Right click and select git bash

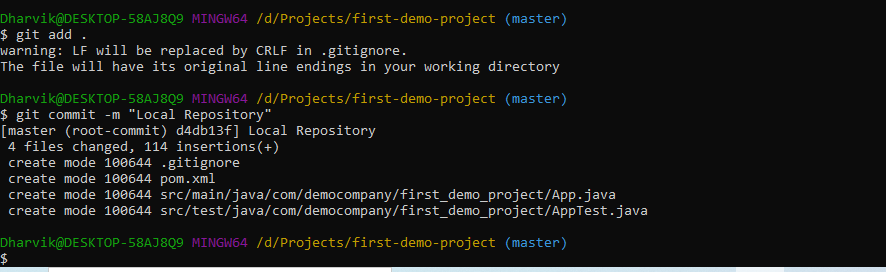


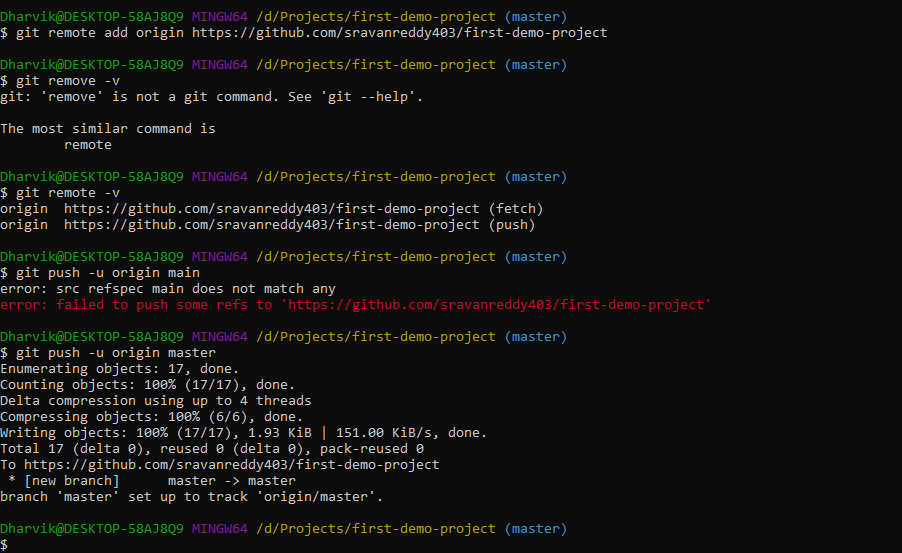






As I no need to push target folder so I need to use gitignore command



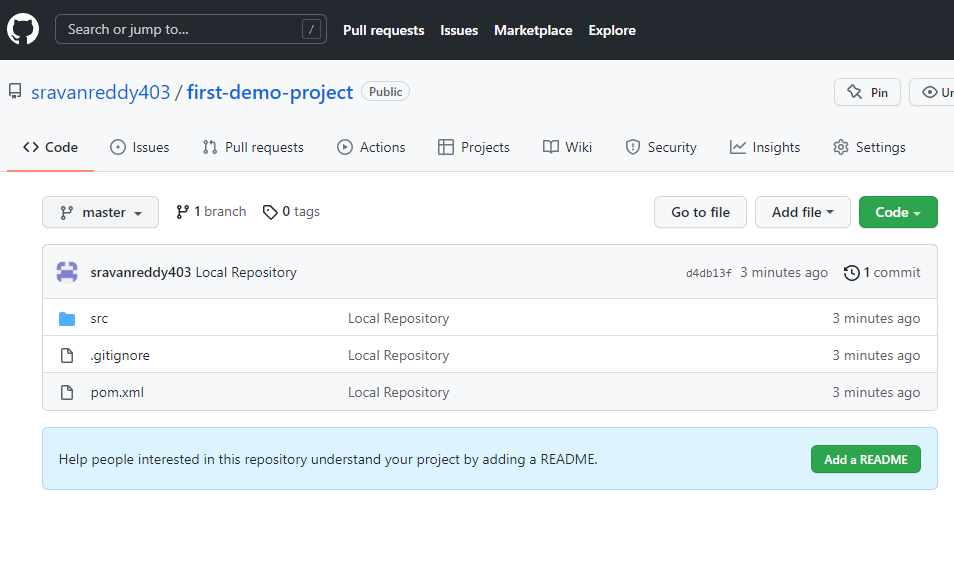


To push the code first add remote

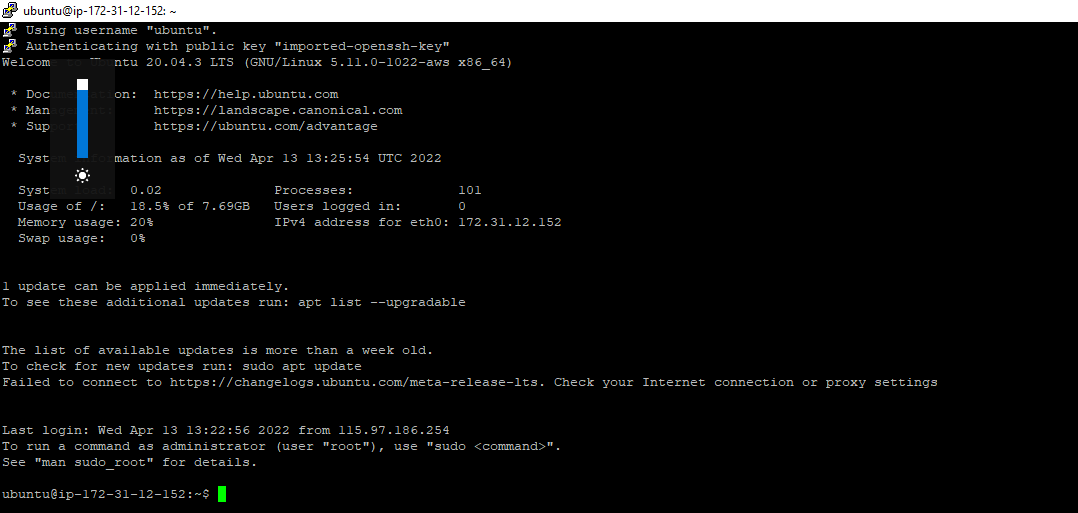
**Git remote add origin “github url”**

**Then**

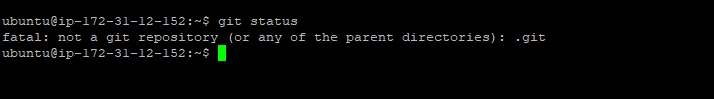
**Git push –u origin master**



**Now launch Ubuntu Linux from AWS EC2 via Putty**

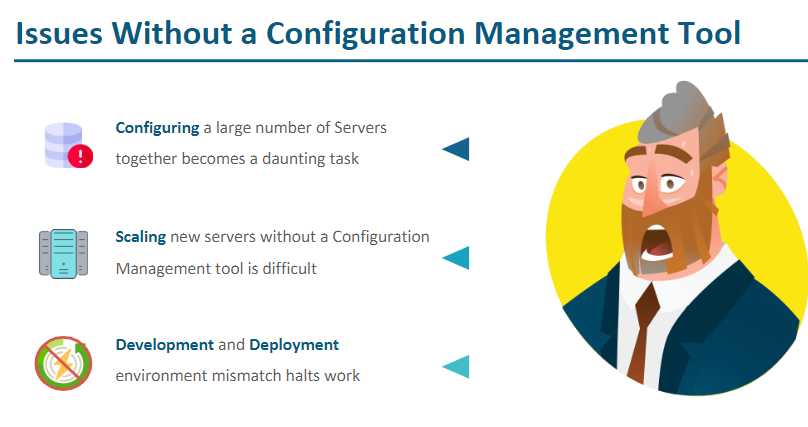


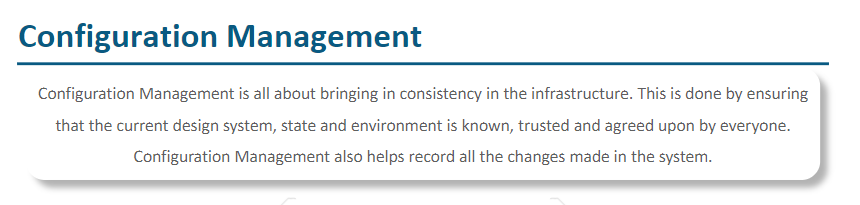
Linux Ubuntu is launched

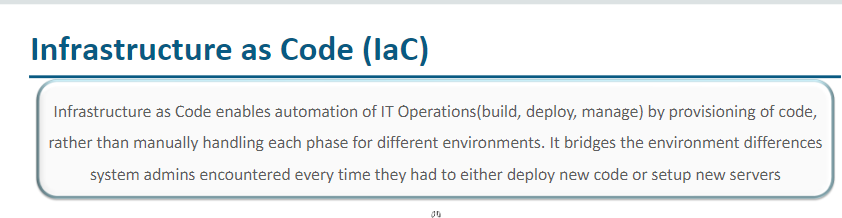


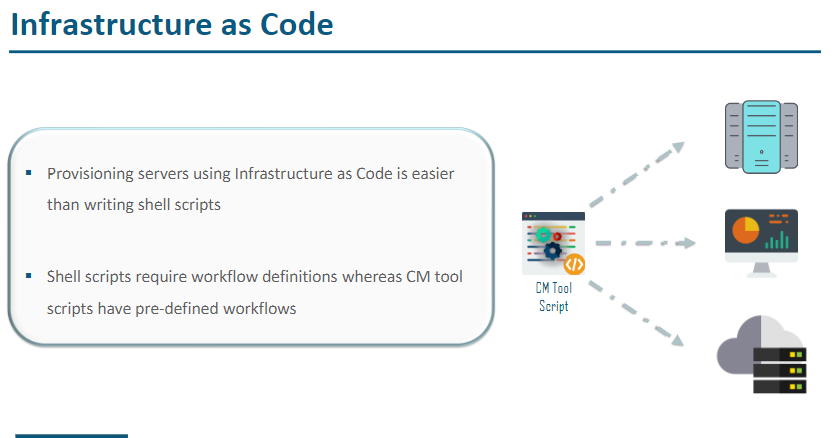
By default git is installed

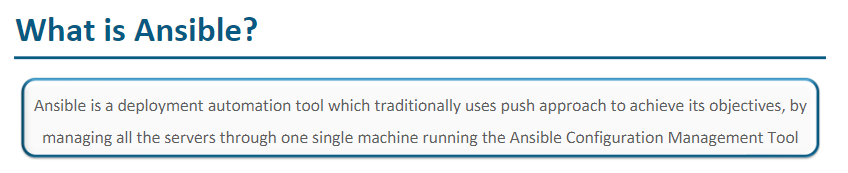
**Ansible:**

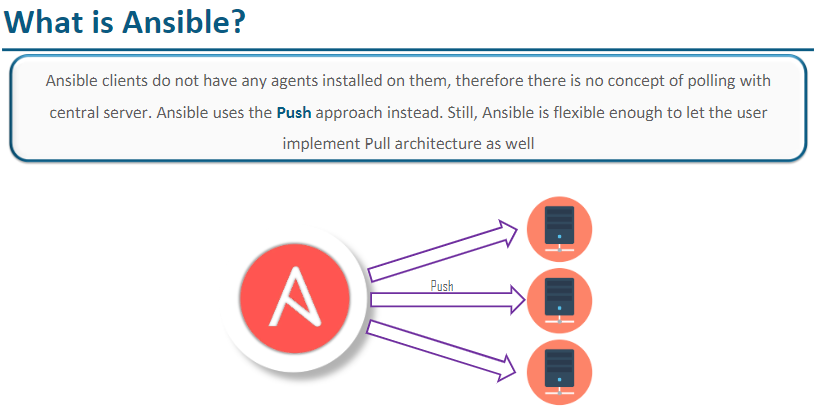


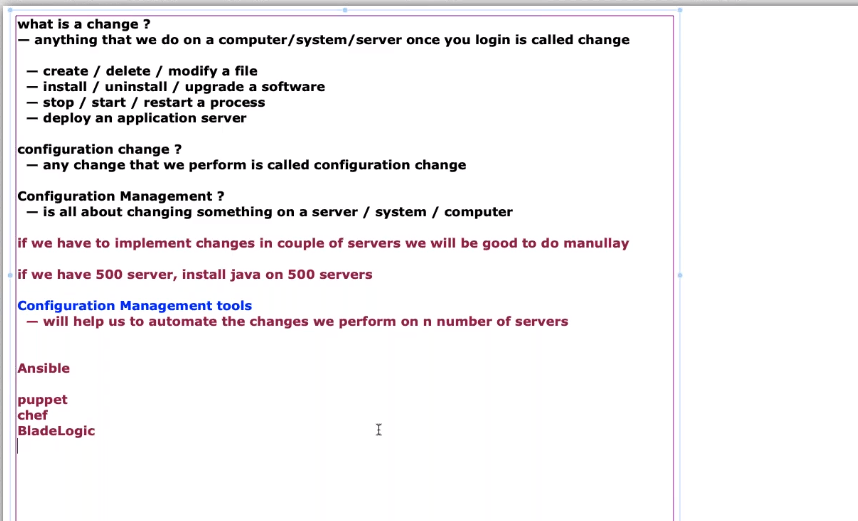


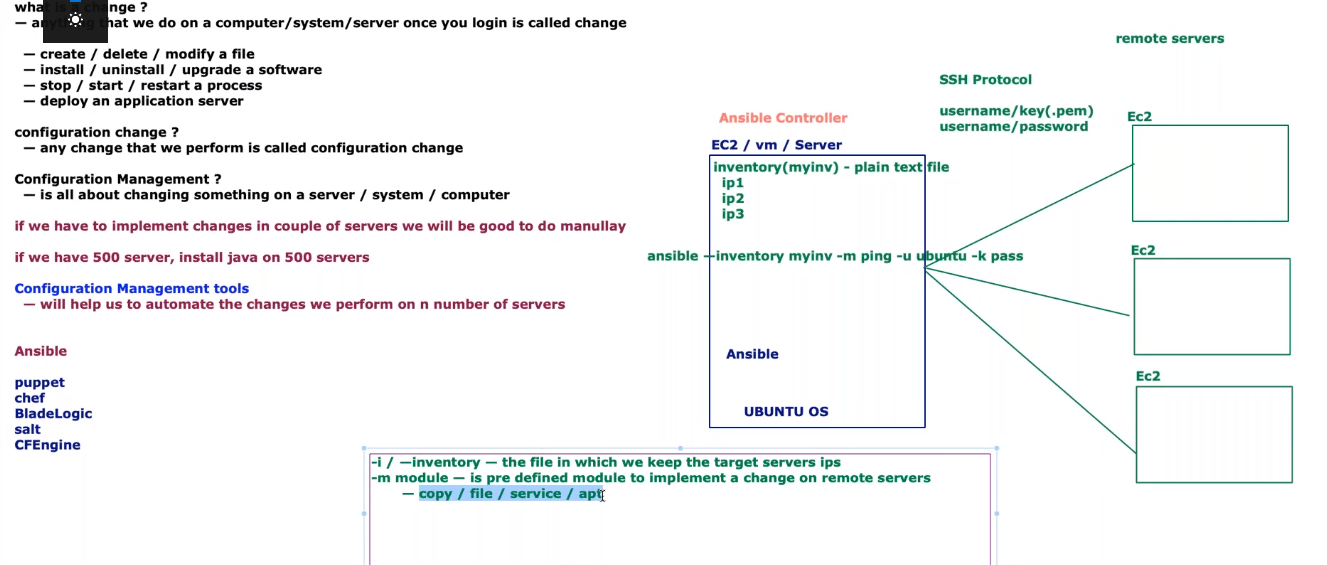


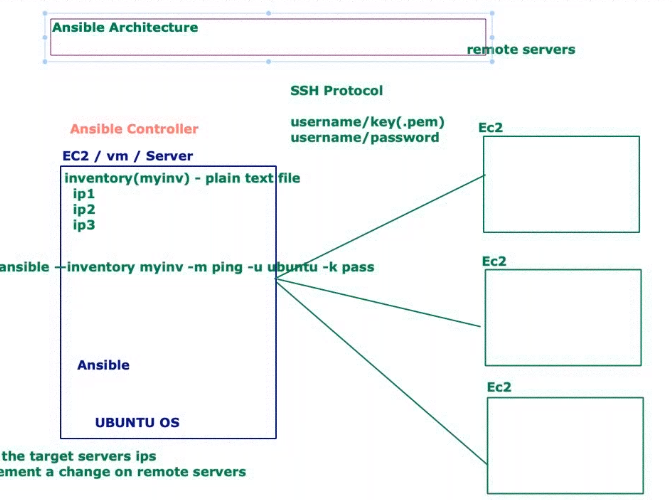


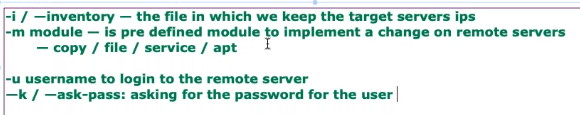












It is recommended to use common user name and common password for all remote servers.

## **Things to know before working with ansible**

1. ansible is agentless ( means, we do not need to run any agent on remote servers for communication )
2. ansible follows push model ( means we push the configuration from ansible controller to remote servers )
3. ansible uses SSH protocol to communicate with remote servers
4. ansible uses below methods for remote servers authentication
   * password based
   * key based

## **Ansible controller ( is the machine/host/server where we install ansible )**

* **/etc/ansible default directory ( we have two important file in it )**
* **hosts ( default inventory file in which we can mention remote servers we want to apply/push the changes to)**
* **ansible.cfg ( default ansible configuraton - can be used to customize as per our need )**

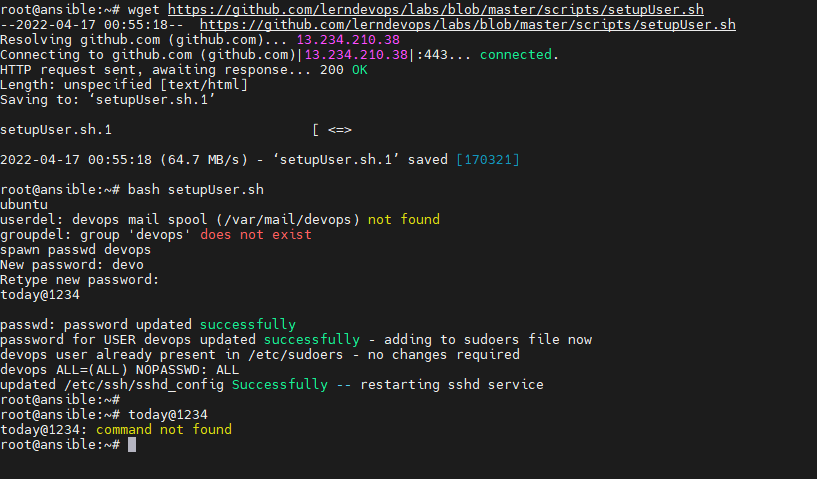
## **Ansible Terminlogy**

1. ansible ad-hoc commands ( single task that can be executed against remote servers )
2. ansible playbooks ( set of instructions written in a .yml file to execute against remote servers )
3. ansible facts ( information about remote servers collected ansible before running any task )
4. ansible modules ( are the units of work that ansible ships to remote servers for executing the tasks )
5. ansible role ( to break a playbook into multiple files. This simplifies writing complex playbooks, and it makes them easier to reuse )

**Ansible execution**

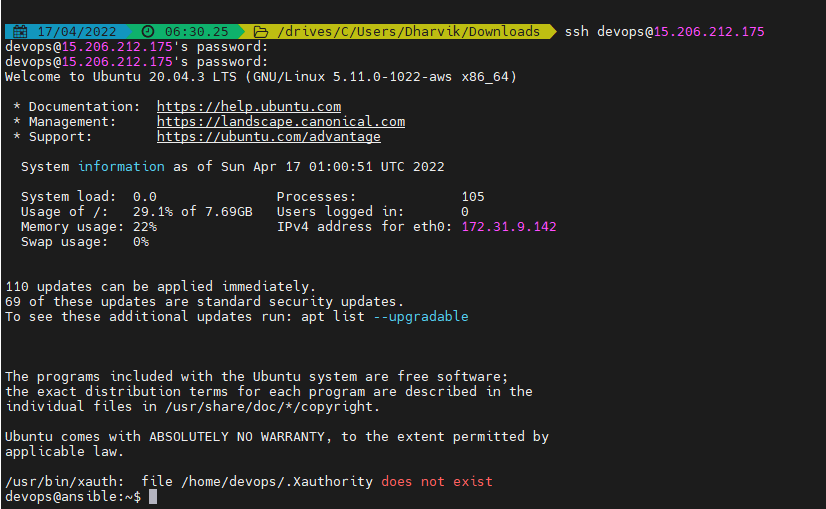
* 1. First create three new VMs and name them as Ansible, Remote1 and Remote2.
  2. Once created do the user setup by running below wget command which will create user name and password for VM login

wget[**https://github.com/lerndevops/labs/blob/master/scripts/setupUser.sh**](https://github.com/lerndevops/labs/blob/master/scripts/setupUser.sh)



* 1. Give full permission
     + chmod 755 setupuser.sh
  2. bash setupUser.sh

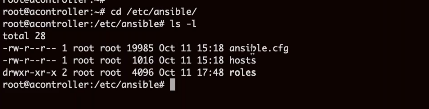
After this we can login to linux using user id and password



* 1. Now install Ansible using below commands on Ansible controller and do not install on remote servers
     + - sudo apt-get update
       - sudo apt-get install software-properties-common
     + sudo apt-add-repository --yes --update ppa:ansible/ansible
     + sudo apt-get install ansible

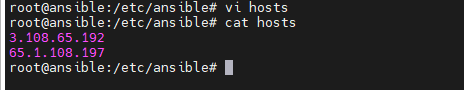
Once installed then we can see ansible status by running **ansible –version**

**Path : /etc/ansible**



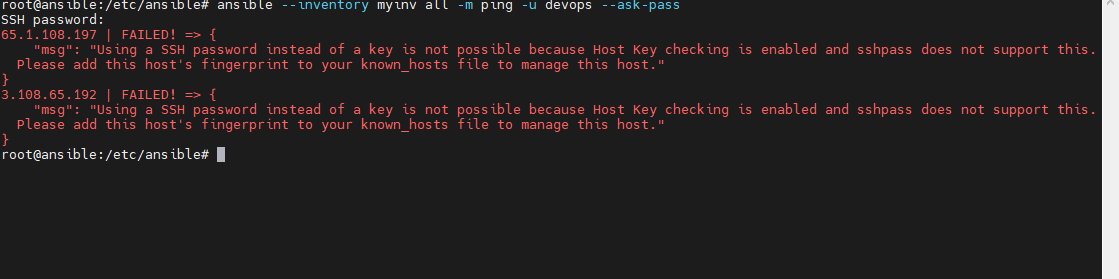
In this path, hosts is default inventory file but we can create our own inventory file.

For now I am adding ip addres of remote servers in hosts file (vi hosts)

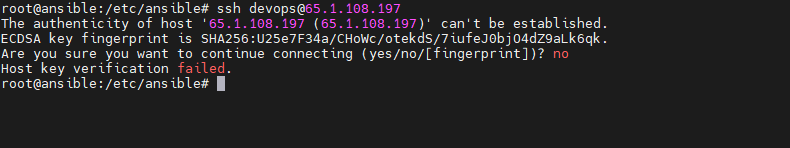


Now run below command to ping the remote servers

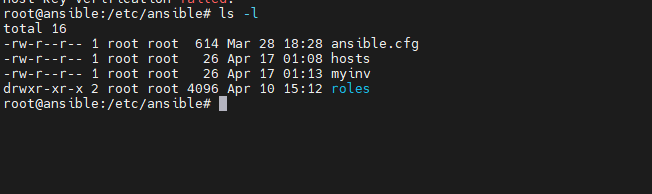
**Ansible –inventory hosts all –m ping –u deops –ask-pass**



Above message is due to when we trying to login remote server by default SSH connection is consideres as No

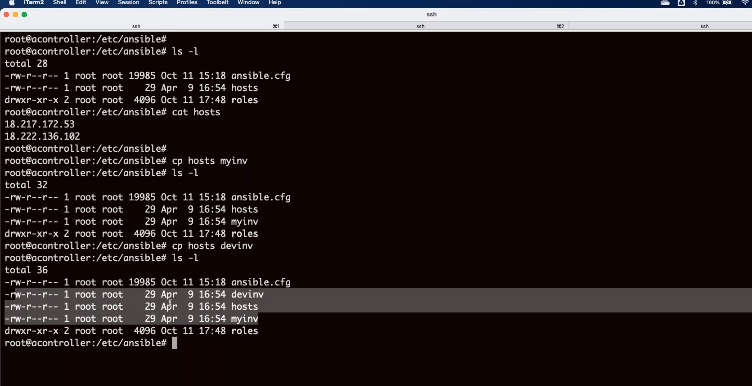


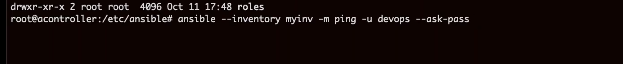
To correct this we need to change hostkey checking as False (Uncomment) in ansible.cfg file

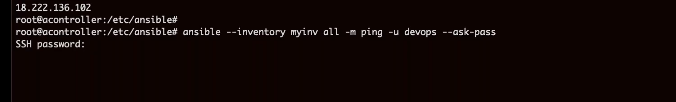


Vi ansible.cfg

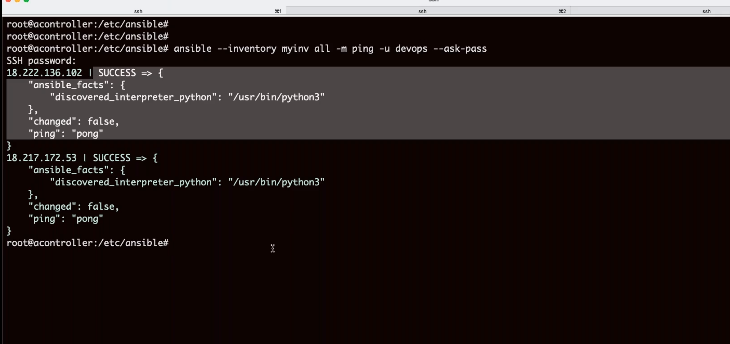
ansible-config init --disabled all > ansible.cfg

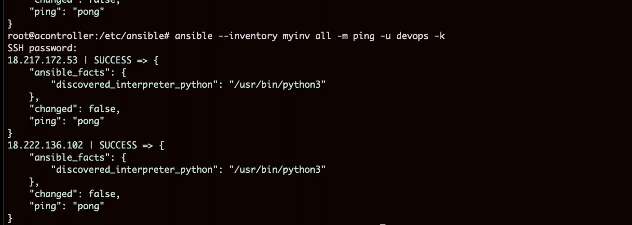






**Modify ansible.cfg (change host key changing to false)**

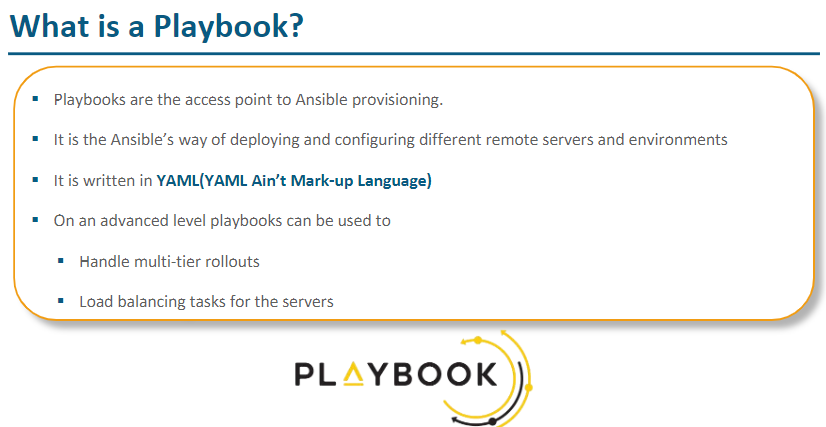


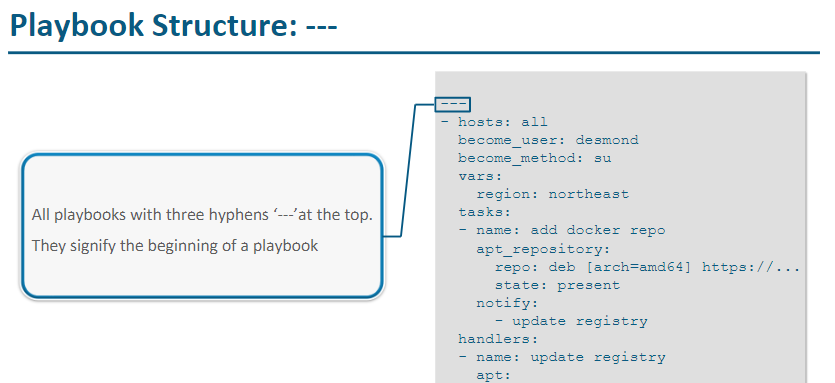


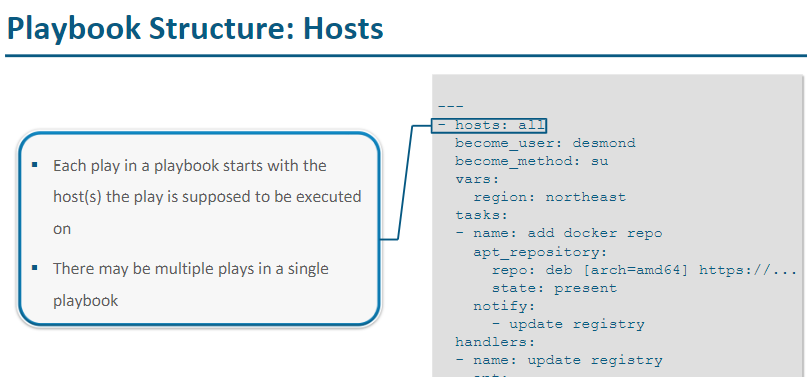
Anisble –I myinv all –m ping –u devops –k

For copy : copy module helps to copy the files into target server



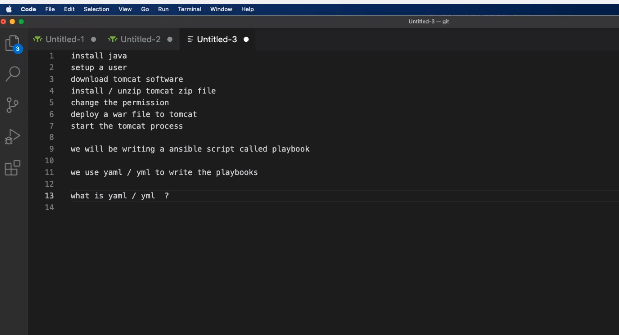






Yaml is everywhere in the devops to write ansible palybooks, docker scripts, kubernetes automation scripts

Yaml ain’t markup language - data serialisation lagauge



Yaml is superset of Json

One rule for yaml is indentation (using spaces)

Yaml is space sensitive and case senisitive

Three data structures in yaml are

Scalars

Dict / map

List /array

1. **Jenkins freestyle project and other types like pipeline, multibranch pipeline?**

**/usr/lib/jvm/java-8-openjdk-amd64**

**/opt/apache-maven-3.8.5**

**Groovy**

**Yaml**

**YAML for Ansible Playbooks**



